Evaluation of the OGF GridRPC Data Management library, and study of its integration into an International Sparse Linear Algebra Expert System

Yves CANIOU[*†]
Université de Lyon, CNRS, JFLI University of Tokyo
E-mail: Yves.Caniou@ens-lyon.fr

Ronan GUIVARCH
Université de Toulouse, INPT ENSEEIHT
E-mail: Ronan.Guivarch@enseeiht.fr

Gaël Le MAHEC
Université de Picardie Jules Verne
E-mail: Gael.Le.Mahec@u-picardie.fr

The Data Management API for the GridRPC describes an optional API that extends the GridRPC standard. It provides a minimal subset of functions to handle a large set of data operations, among which movement, replication, migration and stickyness. We already showed that its use leads to 1) reduced time to completion of application, since useless transfers are avoided; 2) improved feasibility of some computations, depending on the availability of services and/or storage space constraints; 3) complete code portability between two GridRPC middleware; and 4) seamless interoperability, in our example between the French GridRPC middleware DIET and the Japanese middleware Ninf, distributed on French and Japanese administrative domains respectively, leading to both of them contributing to the same calculus, their respective servers sharing only data through our implementation of the GridRPC DM API.

We have extended the implementation of the library and a further integration has been made available into DIET as a back-end of its data manager Dagda. We thus present how the library is used in the International Sparse Linear Algebra Expert System GridTLSE which manages entire expertises for the user, including data transfers, tasks executions, and graphical charts, to help analysing the overall execution. GridTLSE relies on DIET to distribute computations and thus can benefit from the persistency functionalities to provide scientists with faster results when their expertises require the same input matrices. In addition, with the possibility for two middleware to interact in a seamless way as long as they're using an implementation of the GridRPC Data Management API, new architecture of different domains can easily be integrated to the expert system and thus helps the linear algebra community.

The International Symposium on Grids and Clouds (ISGC) 2013
17 - 22 March 2013
Academia Sinica, Taipei, Taiwan

[*]Speaker.

[†]This work is partially founded by the Technological Development Department (D2T) at INRIA, the ANR-JST FP3C (Framework and Programming for Post Petascale Computing) and the ANR-09-COSI-001, COOP (Multi Level Cooperative Resource Management)

1. Introduction

The GridRPC API [13] was designed to define Remote Procedure Call over the Grid. It has been widely and successfully used in middleware like DIET [8], NetSolve/GridSolve [18], Ninf [16], OmniRPC [15] and SmartGridSolve [4]. The API concentrated on remote service executions: It standardized synchronous and asynchronous calls and computing tasks management. In June 2011, the Open Grid Forum standardized the document "Data Management API within the GridRPC" [6] which describes an API that extends the GridRPC standard [13] (GridRPC DM API). Used in a GridRPC middleware, it provides a minimal set of structure definitions and functions to handle a large set of data operations, among which movement, replication, migration, and persistence.

A first prototype of the API containing the basic functions offering data management through the library has been developed, in addition to a GridRPC layer making possible to write a single GridRPC client able to invoke DIET and Ninf services transparently. With several experiments relying on these early developments, we showed in [7] that the GridRPC DM API already answers most of the needs presented in [12], namely **code portability** (the exact GridRPC code written with the GridRPC DM API to manage data can be used with any GridRPC middleware), **computational feasibility** (because of the transparent use of remote data, a GridRPC client is not required to have data on its own system to respect the GridRPC paradigm, hence leading to the use of light client machines), **performance** (useless transfers can be avoided with the use of persistence). Moreover, we showed that it is even possible for two Grid middleware, managing resources across different administrative domains and using the API, to collaborate in a completely transparent manner.

Although the implementation is still in progress, we show in this paper how a sparse linear algebra expert system, GridTLSE[1], can benefit from the use of the data management provided by the API: we first study the possible performance improvements that can be achieved since expertises may use several times the same matrices; then we study how using stickyness for temporary results can lead to spare already work made in a factorization process to perform several remote solves.

In sections 2 and 3, we first recall the need for such an API for the GridRPC and summarize its properties. Then we present the different middleware, context of the use of the library, in Section 4. The different experiments, experimental protocols and results, are detailed in Section 5 and we conclude in Section 6.

2. State of the Art

In the GridRPC paradigm, input and output data are arguments of `grpc_call()` and `grpc_call_async()` and are transferred between clients and servers during steps (4) and (5) of Figure 1. Thus, each Grid middleware managing its own built-in data, **code portability** is impossible, and **performance** can only be managed bypassing the GridRPC model, even for request sequencing [3]. Many other issues arise [5], but they can only be addressed separately: one can store data on distributed file system like GlusterFS[2] or GFarm [17] to deal with automatic replication; OmniRPC introduced omniStorage [14] as a Data Management layer relying on several Data Managers

[1] `http://gridtlse.org`

[2] `http://www.gluster.org/`

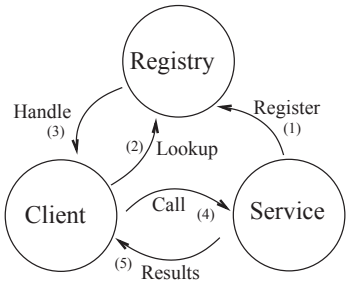

Figure 1: GridRPC paradigm

such as GFarm and Bittorrent. It aims to provide data sharing patterns (worker to worker, broadcast and all-exchange) to optimize communications between a set of resources, but needs knowledge on the topology and middleware deployment to be useful; DIET also introduced its own data managers (JuxMEM, DTM, and DAGDA [2, 9, 10]), which focus on both user data management and persistence of data across the resources, with transparent migrations and replications.

Overall, in addition to only fulfilling complementary services, the lack of standard makes their implementation and usage *not interoperable nor portable* through different middleware, thus the need for the GridRPC DM API standard.

3. GridRPC Data Management API

The GridRPC Data Management standard relies on the definition of 12 functions and of the GridRPC data, which represents a Grid data, *i.e.,* several copies of the same data stored in different locations, and which may store a data in addition to meta-information. Among these, one can cite the **dimension** of the data (vector, matrix or more complicated structures), the **data type** to be able to make the relation with the language type (double, integer, etc). A new and special type, `CONTAINER_OF_GRPC_DATA`, has been introduced as a set where the user can put or get GridRPC data, and can be used with services producing a number of parameters unknown before execution. There is also a **management mode** for each data: *strictly volatile* (the system must erase the data from the storage resource once it has been used or transfered), volatile (the default behavior of the GridRPC standard), *sticky* (a copy of the data must be maintained on the given location), *unique sticky* (in addition to being maintained on the given location, that data must stay the unique copy in the system: some systems may indeed implement some fault tolerance mechanisms and would otherwise replicate the data, a behavior that some users do not want), or *persistent* (the user relies on the GridRPC middleware and its data manager to handle in a seamless and best possible way data management, with possible inner coupling with scheduling decisions). Finally, URI being a description of both the location and the protocol that must be used to access a data registered in the system, one can thus find **two lists of URIs**, describing where each data can be transfered from or has to be transfered to, and **a list of management modes**, each of which being applied to the corresponding data whose URI is given in the output URI list.

The API is driven by 12 functions: **grpc_data_init()** initializes a GridRPC data with information describing the previously mentioned characteristics. GridRPC data referencing input parame-

ters must be initialized with identified data before being used in a `grpc_call()`.
grpc_data_getinfo() lets the user access information like transfer status, location, etc. **grpc_data_transfer()** writes data to output locations (additional ones, with their corresponding management mode, can be given) from input locations (a list of additional input URIs can also be provided). Consequently, some broadcast/multicast mechanisms can then be implemented in the GridRPC data middleware in order to improve performance. **grpc_unbind_data()** may be used when a client does not need the handle on the GridRPC data anymore, but to explicitly erase a data on storage resources and free the GridRPC data, a call to **grpc_free_data()** must be performed. In order to communicate a Grid data between grid users, the GridRPC data management API proposes two functions, **grpc_data_load()** and **grpc_data_save()**. The last functions of the API are related to inner management, to be able to address in-memory data from an URI, and to get and put GridRPC data in a container of GridRPC data, as mentioned previously.

This work relies on the implementation of the GridRPC standard, and evaluates its integration, and its performance both in terms of possibilities and measurements. The library is freely available to download at `https://forge.mis.u-picardie.fr/projects/gridrpcdm/`. Note that studies to integrate popular data protocols like iRods[3] and OwnCloud[4] are in progress.

4. Integrating GridRPC Data Management into GridTLSE

4.1 GridTLSE

The expert site GridTLSE[5] for linear algebra aims at providing tools and softwares for sparse matrices (matrices with a higher ratio of zero components versus nonzero ones). The site provides user assistance to evaluate and choose the best solver for given problems and helps to set the appropriate values of the input parameters that control the efficiency of the selected solver (Sparse Direct Solvers currently available in GridTLSE are MA48, MA49, MUMPS, SuperLU, UMFPack).

The GridTLSE project uses a high level component semantic description to manipulate sparse linear algebra services (factorization, orderings, linear solves, etc.). Figure 2 presents the different layers of the GridTLSE platform: to provide an easy access to these services, GridTLSE uses the scenario concept which is a graphical description of the task workflow to perform (*Geos*). This description relies on a semantic description of sparse linear algebra services and tools (*Prune*). The expertise engine (*Weaver*) takes into account the user requirements, the internal expertise scenarios, the constraints on the solvers to build a dynamic experience workflow that we call expertise. Finally, each experience of this workflow corresponds to an execution of a computation service, which is made through calls to the DIET middleware (with the help of the *GridCOM* grid adaptor).

4.2 DIET, a GridRPC framework

DIET [8] (**D**istributed **I**nteractive **E**ngineering **T**oolbox) is a lightweight GridRPC middleware, designed for high performance. It is highly scalable, a deployment on thousands of nodes requering only a few minutes. It integrates many features, like customizable distributed scheduling, and can

[3] `https://www.irods.org/`
[4] `https://owncloud.org/`
[5] `http://gridtlse.org`

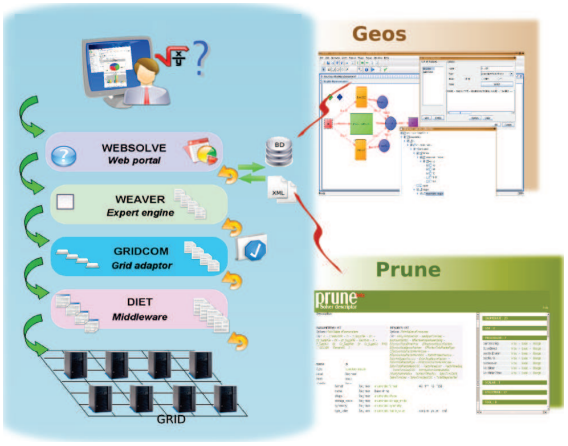

Figure 2: GridTLSE platform

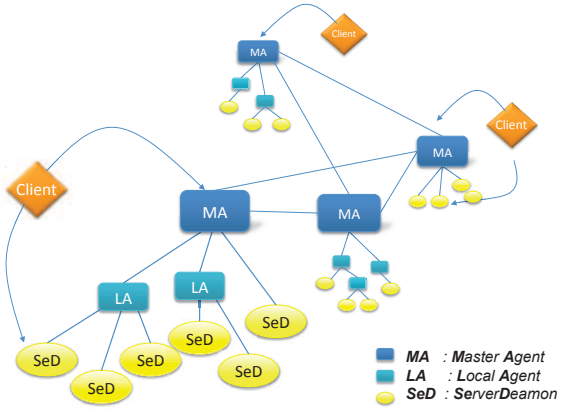

Figure 3: A DIET hierarchy

rely on dynamic workflow management, as well as LRMS and Cloud seamless management. It is implemented in CORBA and thus benefits from the many standardized, stable services provided by freely-available and high performance CORBA implementations. It comprises several components: A **Client** uses the DIET infrastructure to solve problems using a RPC approach. A **SED** (server daemon) acts as the service provider, exporting functionalities via a standardized computational service interface; a single SED can offer any number of computational services. A SED can also serve as the interface and execution mechanism for either a stand-alone interactive machine or a parallel supercomputer, by directly and independently dealing with its batch scheduling facility. The third component of the DIET architecture, the **agent**, facilitates the service location and invocation interactions of clients and SEDs. The DIET deployment is structured hierarchically for improved scalability, the hierarchy of agents being composed of a single **Master Agent (MA)** (the entry point) and several **Local Agents (LA)**, providing higher-level services such as scheduling and data management. Figure 3 shows an example of a multi-hierarchy DIET architecture.

DIET relies on DAGDA [10] to perform its data management. DAGDA can make implicit and explicit replication, can optimize transfers by selection the most convenient source. For this work, we implemented a "glue" for DAGDA to use the GridRPC data management library.

5. Experimentations and results

5.1 Platform of experimentation

Depicted in Figure 4, the testbed is composed of the machine running the GridTLSE web site (and its matrix collection) in Toulouse, and a cluster in Lyon, with the MA and the client launched on the frontal, and two SEDs running on two of its computing resources.

5.2 Validation of the integration of data management in GridTLSE

5.2.1 Methodology

Each expertise uses one of the 5 matrices given in Table 1, with their respective raw and compressed sizes. For each expertise, we consider 3 test cases, each containing two calls to a

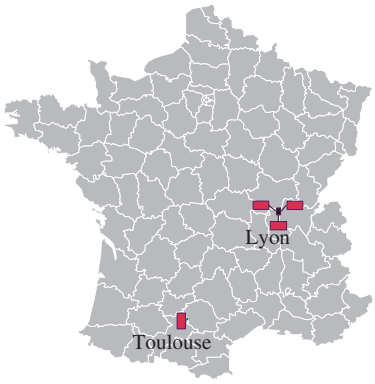

Figure 4: A geographically distributed platform of experimentation

	Aster_feti009a_2.1		*Aster_perf001a*		*NICE20MC*		*QIMONDA07*		*FLUX − 2M*	
	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time
raw	1 563	-	534 224	-	895 087	-	2 195 699	-	16 007 994	-
gziped	353	0	131 813	13	205 756	20	214 549	34	3 000 860	325

Table 1: Matrices characteristics and time to decompress (sizes in KBytes, and times in seconds)

solver requesting the use of the same matrix. In the following, we give the overall duration of the transfers of each test case, using M and Mc to name the same raw and compressed matrix, T_M the duration of the transfer of matrix M, d the duration of the decompression process, B_1 bandwidth between the machines in Lyon and the GridTLSE website, B_2 between the machines in the cluster.

Test case 0 Do not use DAGDA. Transfers are explicitly made: when a service is executed on a SED, it downloads the gzipped matrix from the GridTLSE website. Hence, two transfers occur and $D_0 = (T_{Mc}/B_1 + d) * 2$

Test case 1 Use DAGDA. The client downloads the gzipped matrix from the GridTLSE site, uncompresses and registers it into DAGDA by sending the uncompressed matrix to the MA. Then the SED downloads the matrix from the MA. When the second call is performed on the same SED, there is still a copy of the matrix on the SED, so the data is immediately available and $D_1 = (T_{Mc}/B_1 + d) + T_M/B_2$

Test case 2 Use DAGDA. The expertise is the same than for Test cases 0 and 1, but the second execution is conducted on a different computing resource, thus an additional transfer from the MA (but more generally DAGDA is able to choose the best location from where to download data depending on internal statistics and monitoring). Hence we have $D_2 = (T_{Mc}/B_1 + d) + 2 * T_M/B_2$

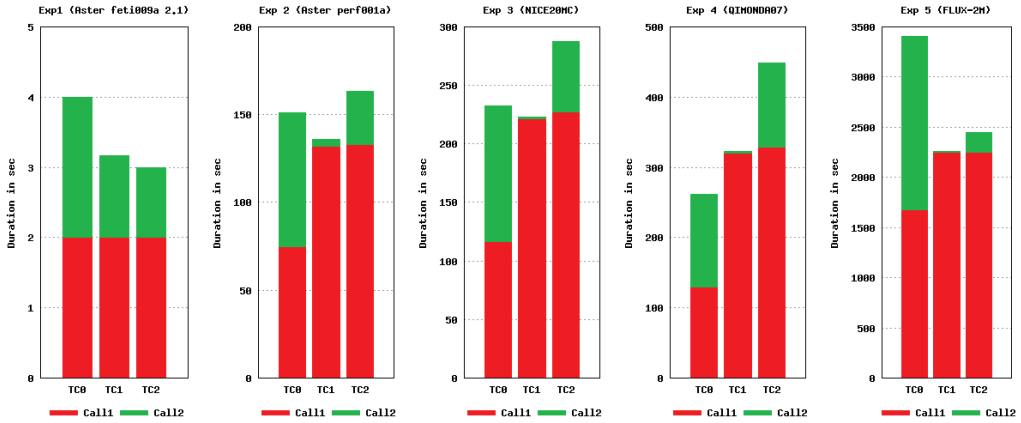

Figure 5: Results for each expertise on all the test cases

5.2.2 Results

Apart from `Aster_feti009a_2.1` whose timings are too small to be really meaningful, results depicted in Figure 5 are really interesting. Similar experiments were conducted in [11], but results are slightly different here, because of a network bandwidth whose performance has increased and DAGDA which relies on the GridRPC data management library. Indeed, apart from the expertise using `FLUX-2M`, for which DAGDA shows a real improvement as long as a matrix is used at least twice in an expertise, we can see that the cost to register a matrix in the data management layer is higher than expected: for `Aster_perf001a` and `NICE20MC` the expertise (2 GridRPC calls) finishes in the same range of time for TC0, TC1 and TC2, but actually the two calls for TC1 finish nearly at the same time. Of course, TC2 involving an additional local transfer, it conducts to a slightly worse time to complete the expertise. For `QIMONDA07`, as long as fewer expertises use a matrix less than 3 times, direct download from the GridTLSE website gives a smaller expertise duration.

5.2.3 Remarks

Overall the results presented in the previous section can be considered as **the minimum gain** that can be achieved for an expertise. Indeed 1) our cases do not consider the fact that several expertises can be performed in parallel, and thus transfers are made in parallel, sharing the network resource and taking more time; 2) if expertises are executed on the same parallel machine, and use the same matrix (or matrices), there is only one transfer using DAGDA, improving by far the time to completion of the study; 3) DAGDA smartly manages storage memory (in a Least Recently Used – LRU – or Least Frequently Used manner – LFU –), so that matrices are still available locally as long as the platform is deployed and there is still memory available; 4) in this study, a matrix is only used twice: in a complex expertise analysis, where a scientist wants to study the effect of some scheduling parameters, gains are cumulative. Hence negative results such as for the expertise using `QIMONDA07` would rarely happen and could be easily avoided, by storing both compressed and non-compressed data; 5) finally, the issue for `QIMONDA07` being known, we will improve the solution with additional work in order to avoid the transfers from and to the MA.

Moreover, as shown in [7], more GridRPC-driven architectures may be available in a seamless way with all the benefits of the implemented solution. This leads to good expectations on a wider platform (French-Japanese for example) since using DAGDA leads to automatic, yet transparent of use, replicates on components of the DAGDA hierarchy.

5.3 Improving GridTLSE expertises with the use of factor files

We consider now a second example that manipulates another kind of data: the factor files resulting from the factorization of a matrix.

5.3.1 Methodology

Numerical context

A typical direct solver for solving sparse linear system $Ax = b$ consists of 3 different steps:

1. a symbolic factorization that determines the nonzeros structures of the factors L and U
2. a numerical factorization that computes the factors L and U such that $L.U = A$
3. a solve step

One can see that it is interesting to separate steps 1 and 2 from step 3: for an unique factorization, we can perform multiple solves with different values of the right hand side (RHS) b.

This situation arises in many scientific fields when there are multiple systems to solve with the same matrix either in a simultaneous (parallel) or sequential way. For instance, different systems are solved to evaluated a solution field, function of a wave frequency: for each selected frequency, there is a different RHS. Alternatively, in an evolution problem where, at each time step k, there is a system $Ax_k = b_k$ to solve.

Therefore if we are able to save the factors after the factorization steps, we can have two separate pieces of software: one that performs the factorization and dumps the factor on disk; one that loads the factor and performs a solve.

Not all direct solvers offer the dump/load functionalities that are able to manage big matrices. The forthcoming version 5.0 of MUMPS [1] enables us to write such pieces of software.

Implementation in GridTLSE

We can describe the computation to perform a complete direct solve as a workflow. Figure 6 shows the workflow implemented in GridTLSE.

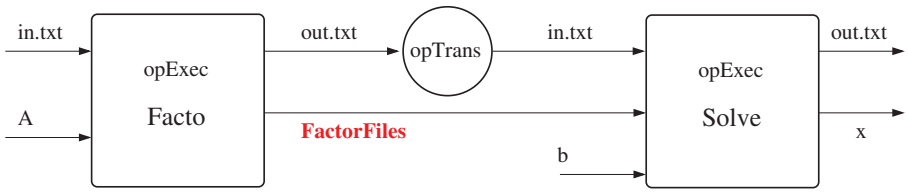

Figure 6: A direct solve as a workflow in GridTLSE

In the previous version (called *http version*), the factor files are uploaded to the GridTLSE website by the DIET SeD that performs the factorization. The SeD that computes the solution downloads these files at the beginning of its work.

	twotone		*Bmw*3_2		*human_gene*1	
	Size	Time	Size	Time	Size	Time
raw	26509	-	132765	-	378890	-
gziped	5434	0.4	35113	3	135785	10

Table 2: Matrices characteristics and time to decompress (sizes in KBytes, and times in seconds)

The new version (called *dagda version*) uses DAGDA. The factor files are registered into DAGDA at the end of the factorization step with no upload to the website. If the solve is performed on the same cluster or even better, on the same machine, it can rapidly retrieve the factor files.

5.3.2 Results

We perform different experiments with the selection of matrices detailed in Table 2; the size of the resulting factors is the main parameter to show the gain of time when we avoid the communication of the factor files between the cluster and the GridTLSE website. Table 3 presents the obtained results with or without compression and with or without advanced data management features. For these experiments, we used the test cases TC0, TC1 and TC2 of Section 5.2.1. We can clearly notice the benefit of using the GridRPC Data Management library through DAGDA when we look at the **total time**[6].

Concerning the different compressions, when we are using the *http version*, we can say that *gzip* compression gives the best results: the smaller time in transfer due the reduced size with *lzma* does not compensate the larger time spent in the compression and decompression steps.

[6] the time of all steps are not given

Matrix	Matrix Size	Factor Size	Compression / Decompression Times	Up./Down. SuppFiles Times	Facto. Time	Solve Time	Total Time
twotone							
TC0	5.56MB	394MB	- / -	37 / 204 s.	38.46 s.	1.45 s.	305 s.
TC0 (gzip)		160MB	27 / 5 s.	17 / 82 s.	38.28 s.	1.44 s.	200 s.
TC0 (lzma)		147MB	492 / 39 s.	16 / 76 s.	38.34 s.	1.42 s.	689 s.
TC1		394MB	- / -	0 / 0 s.	38.24 s.	1.44 s.	60 s.
TC2		394MB	- / -	0 / 22 s.	38.24 s.	1.44 s.	105 s.
BMW3 2							
TC0	36MB	695MB	- / -	63 / 356 s.	44.15 s.	3.05 s.	555 s.
TC0 (gzip)		485MB	63 / 12 s.	45 / 242 s.	45.13 s.	3.05 s.	507 s.
TC0 (lzma)		462MB	811 / 117 s.	43 / 232 s.	44.15 s.	3.06 s.	1355 s.
TC1		485MB	- / -	0 / 0 s.	44.39 s.	3.05 s.	124 s.
TC2		485MB	- / -	0 / 39 s.	44.39 s.	3.05 s.	185 s.
human gene1							
TC0	139MB	1689MB	- / -	151 / 761 s.	638 s.	10 s.	2203 s.
TC0 (gzip)		1412MB	183 / 50 s.	126 / 729 s.	630 s.	10 s.	2442 s.
TC0 (lzma)		1356MB	2195 / 345 s.	121 / 613 s.	645 s.	9 s.	4711 s.
TC1		1689MB	- / -	0 / 0 s.	639 s.	11 s.	1361 s.
TC2		1689MB	- / -	0 / 96 s.	639 s.	11 s.	1480 s.

Table 3: Experiments results for three matrices, three test cases and two compression algorithms

6. Conclusion and future work

We have presented the first evaluation of the benefits obtained by the extension of the GridRPC API OGF standard concerning data management, *i.e.,* the GridRPC Data Management GFD-R-P.186 OGF standard, in the context of real computations required for a sparse linear algebra expert system, GridTLSE.

We have conducted several experiments. The experiments for validation showed us that the proposed solution is interesting if the same matrix is used more than a couple of times in an expertise. Of course, in a production environment, some gains would rapidly be obtained since different expertises are generated by GridTLSE, and they may use the same input matrices. But some additional work on the architecture of the solution, *i.e.,* the integration of GridTLSE and DIET, together with the use of the data management, can improve the solution by overlapping given operations and thus reducing the time spent to register a new data in the system.

We also studied the possibility of using the factor files produced by direct solver MUMPS, in order to split the factorization step and the solve step. The experiments clearly demonstrated that the new pieces of software developed for this study perform much better than the basic approach. They should be integrated in the production expert site soon.

Future work will go towards the development of the full API (some functions are still missing), also taking into account other protocols such as `GridFTP`, iRods and OwnCloud if possible. We will continue our work in the Open Grid Forum GridRPC working group: recommendations and possible extensions to the API are possible, but it will depend on the final users'/developers' remarks. We are eager to get different use-cases and users' feedback.

On the numerical side, future work will focus on the two following directions: 1) When a full direct solve expertise is launched, there may be no need to perform the factorization step if it has already been processed before, for the need of another expertise for example. Mechanisms have thus to be deployed inside the expertise engine to detect this situation. For instance, we could verify if the factor files corresponding to the matrix are already registered in the data manager (to avoid the factorization step). This could actually be done at the middleware level or in *Weaver*, the expertise engine of GridTLSE in connection with the DAGDA data manager. 2) Extend our work to the parallel version of the direct solvers. That will induce a more complicated data management than with the sequential version, in addition to automatic job management with regard to LRMS systems.

References

[1] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. SIAM Journal on Matrix Analysis and Applications, 23(1):15–41, 2001.

[2] G. Antoniu, M. Bertier, L. Bougé, E. Caron, F. Desprez, Mathieu Jan, S. Monnet, and P. Sens. GDS: An architecture proposal for a grid data-sharing service. In V. Getov, D. Laforenza, and A. Reinefeld, editors, Future Generation Grids, volume XVIII, CoreGrid Series of Proceedings of the Workshop on Future Generation Grids November 1-5, 2004, Dagstuhl, Germany. Springer Verlag, 2006.

[3] D.C. Arnold, D. Bachmann, and J. Dongarra. Request sequencing: Optimizing communication for the Grid. In EUROPAR: Parallel Processing, 6th International EURO-PAR Conference. LNCS, 2000.

[4] T. Brady, M. Guidolin, and A. Lastovetsky. Experiments with SmartGridSolve: Achieving higher performance by improving the GridRPC model. In The 9th IEEE/ACM ICGC, 2008.

[5] Y. Caniou, E. Caron, G. Le Mahec, and H. Nakada. Standardized data management in GridRPC environments. In 6th International Conference on Computer Sciences and Convergence Information Technology, ICCIT'11, Jeju Island, Korea, Nov. 29 - Dec. 1 2011. IEEE.

[6] Y. Caniou, E. Caron, G. Le Mahec, and H. Nakada. Data management API within the GridRPC. In GFD-R-P.186, June 2011.

[7] Yves Caniou, Eddy Caron, Gaël Le Mahec, and Hidemoto Nakada. Transparent Collaboration of GridRPC Middleware using the OGF Standardized GridRPC Data Management API. In The International Symposium on Grids and Clouds (ISGC), page 12p. Proceedings of Science, February 26 - March 2 2012.

[8] E. Caron and F. Desprez. DIET: A scalable toolbox to build network enabled servers on the grid. In International Journal of High Performance Computing Applications, volume 20(3), pages 335–352, 2006.

[9] B. Del-Fabbro, D. Laiymani, J.M. Nicod, and L. Philippe. DTM: a service for managing data persistency and data replication in network-enabled server environments. Concurrency and Computation: Practice and Experience, 19(16):2125–2140, 2007.

[10] F. Desprez, E. Caron, and G. Le Mahec. DAGDA: Data Arrangement for the Grid and Distributed Applications. In AHEMA 2008. International Workshop on Advances in High-Performance E-Science Middleware and Applications. In conjunction with eScience 2008, pages 680–687, Indianapolis, Indiana, USA, December 2008.

[11] Frédéric Camillo, Yves Caniou, Benjamin Depardon, Ronan Guivarch, and Gaël Le Mahec. Design of an international sparse linear algebra expert system relying on an OGF GridRPC Data Management GridRPC system. In 7th International Conference on Computer Sciences and Convergence Information Technology, ICCIT'12, pages 176–181, Seoul, Korea, dec 2012. IEEE.

[12] Frédéric Camillo, Yves Caniou, Benjamin Depardon, Ronan Guivarch, and Gaël Le Mahec. Improvement of the data management in gridtlse, a sparse linear algebra expert system. JCIT: Journal of Convergence Information Technology, 8(6):562–571, 2013.

[13] H. Nakada, S. Matsuoka, K. Seymour, J.J. Dongarra, C. Lee, and H. Casanova. A GridRPC model and API for end-user applications. In GFD-R.052, GridRPC Working Group, June 2007.

[14] Y. Nakajima, Y. Aida, M. Sato, and O. Tatebe. Performance evaluation of data management layer by data sharing patterns for GridRPC applications. In LNCS Euro-Par 2008 - Parallel Processing, volume 5168, pages 554–564, 2008.

[15] M Sato, M Hirano, Y Tanaka, and S Sekiguchi. OmniRPC: a GridRPC facility for cluster and global computing in OpenMP. OpenMP Shared Memory Parallel Programming, 2104:130–136, 2001.

[16] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, and S. Matsuoka. Ninf-G: A reference implementation of RPC-based programming middleware for grid computing. Journal of Grid Computing, 1(1):41–51, 2003.

[17] O. Tatebe, K. Hiraga, and N. Soda. Gfarm grid file system. New Generation Computing, 28:257–275, 2010.

[18] A. YarKhan, J. Dongarra, and K. Seymour. GridSolve: The evolution of network enabled solver. In James C. T. Pool Patrick Gaffney, editor, Grid-Based Problem Solving Environments: IFIP TC2/WG 2.5 Working Conference on Grid-Based Problem Solving Environments (Prescott, AZ, July 2006), pages 215–226. Springer, 2007.